
CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

TRABAJO FIN DE GRADO

# Sistemas web con alta disponibilidad en cloud

Autor: Julio Chana Moreno

Directora: María Luisa Córdoba Cabeza

MADRID, ENERO DE 2015

*A mi familia y amigos, por estar siempre ahí.*
*A Marisa, por su apoyo todos estos años.*

# Índice

**I Introducción** **1**

**II Resumen** **3**

**1. Español** **3**

**2. English** **3**

**III Estructuras simples vs. sistemas en alta disponibilidad** **5**

**3. Definiciones. Beneficios y costes** **5**

**4. Tipos de alojamiento** **6**

**5. ¿Cuándo se debe usar un sistema en alta disponibilidad?** **6**

**IV *Software*** **9**

**6. Balanceador de carga** **9**

**7. Caché** **10**
7.1. Funcionamiento de peticiones a una caché . . . . . 10

**8. Servidor web** **11**

**9. Sistema gestor de bases de datos** **11**
9.1. Bases de datos relacionales . . . . . 11
9.2. Bases de datos no relacionales . . . . . 11

**10. Respaldo de datos** **11**

**11. DNS** **12**

**V Métodos para conseguir alta disponibilidad y adaptabilidad al tráfico** **13**

**12. Alta disponibilidad a nivel físico** **13**

**13. Alta disponibilidad a nivel lógico** **13**

**14. Adaptabilidad al tráfico** **14**

**15. Replicación de bases de datos** **14**
15.1. Bases de datos relacionales . . . . . 14
15.2. Bases de datos no relacionales . . . . . 15

**VI Monitorización** **17**

**16. Métodos de monitorización** **17**

**17. Cacti: propuesta de *software* de monitorización** **18**
17.1. Obtención de la información . . . . . 18
17.2. Representación . . . . . 18

**VII Guía orientativa para montaje de sistema de alta disponibilidad en servicios *cloud*** **21**

**18. Consideraciones previas** **21**

**19. Sistema propuesto** **23**
19.1. Explicación del sistema propuesto . . . . . 24

**20. Configuraciones de servicios** **26**
20.1. DNS . . . . . 26
20.2. Balanceador . . . . . 26
20.3. Caché . . . . . 28
20.4. Servidor web . . . . . 30
20.5. Bases de datos . . . . . 34
20.6. CDN . . . . . 37
20.7. Respaldo de datos . . . . . 39

**VIII Reflexión personal** **43**

**21. Análisis teórico de beneficios** **43**
21.1. Sistema con servidores propios . . . . . 43
21.2. Sistema con servidores en *cloud* . . . . . 46

**22. Análisis de tráfico de entrada a los servidores web según el tiempo de caché** **47**

**23. Líneas futuras** **49**
23.1. Autoescalado . . . . . 49
23.2. Orquestación . . . . . 50

**24.Conclusiones** **50**

## IX Bibliografía 53

# Parte I
# Introducción

Este trabajo recoge el estudio de las distintas posibilidades existentes por las que se pueden optar a la hora de montar un sistema orientado a web. Profundiza en la preparación para conseguir alta disponibilidad, teniendo en cuenta cuándo puede ser necesario optar por este tipo de tecnologías más avanzadas.

El sistema de alta disponibilidad que se propone será capaz de adaptarse a las necesidades temporales por las que atraviese, esto significa que podrá aumentar o disminuir su número de nodos dependiendo de la necesitad de trabajo en un momento puntual.

Además del estudio de estas tecnologías, se recogen los resultados de una implantación a pequeña escala de un sistema de pruebas, mediante el cual se puede analizar el comportamiento de las tecnologías utilizadas.

Es necesario comentar que no se busca un resultado específico y único, puesto que no existe el sistema perfecto para todas las necesidades posibles. Es por ello que se ha seleccionado un caso específico y se ha elaborado un procedimiento a modo de guía para explicar paso a paso cómo montar un sistema con las características mencionadas.

# Parte II
# Resumen

## 1. Español

Este trabajo contiene el estudio de las tecnologías que se están usando actualmente en web, tratando de explicar cuáles son sus principales componentes, su objetivo y funcionamiento.

En base a un supuesto teórico de un montaje para un servicio web con un número muy alto de usuarios, y basándose en las tecnologías estudiadas, se propone un posible montaje completo de un sistema, que sería capaz de gestionar correctamente todas las peticiones, evitando fallos y tiempos de indisponibilidad.

Se añade un análisis teórico de los costes deribados de la implantación del sistema, comparándolo con un sistema web convencional, y otro análisis con el funcionamiento de una caché y los beneficos, en carga, derivados de su uso.

## 2. English

This work contains a study about new web technologies. Its objective is to explain the web technologies componentes with their particular usage and performance.

Based on a theorical postulation about a preparation of a web service with a large number of users, and working with the studied technologies, a complete system assembling is proposed. This system will be able to attend all the incoming requests, without failures nor downtimes.

It is attached a theorical study of the derivative costs associated to the system implementation, compared to a traditional one. In addition, another study is included with the workflow of a cache and the benefits derived of its usage in work terms.

# Parte III
# Estructuras simples vs. sistemas en alta disponibilidad

## 3. Definiciones. Beneficios y costes

Para el contexto de este trabajo, se puede definir un sistema web como un servicio públicamente accesible que sea capaz de recibir peticiones HTTP, gestionarlas, y proporcionar respuesta al solicitante.

Cuando se habla de una estructura simple, es porque detrás de ese sistema web abstracto, se encuentra uno o varios servidores que no tienen la capacidad de adaptarse a las necesidades por las que atraviesa el sistema, y que, por tanto, son susceptibles de sufrir sobrecargas que lleven a la interrupción del servicio.

Por otra parte, cuando se habla de sistemas en alta disponibilidad, es porque el sistema web es capaz de adaptarse a la carga, realizar gestión del tráfico para no sobrecargar los nodos con más cantidad de trabajo y asignar las peticiones a aquellos que estén libres. Además, son capaces de aumentar su tamaño y, de este modo, redistribuir la carga total del sistema provocando una bajada de la carga puntual en cada nodo.

Es importante tener en cuenta el coste de ambos sistemas. Generalmente se piensa que los sistemas simples tienen un coste menor, puesto que suele ser más barata su implantación y mantenimiento, además de que suponen un coste fijo y conocido.

El problema que supone esta afirmación es que no se tienen en cuenta los costes derivados de la interrupción del servicio, que pueden darse si el sistema sufre un fallo interno (hardware o software independiente al servicio). Es muy probable que estos costes (o falta de ingresos), sean mucho mayores que el coste derivado de gestionar un sistema en alta disponibilidad, que contempla estos posibles fallos, minimizándolos casi por completo.

En los sistemas de alta disponibilidad, que son adaptables al uso puntual a lo largo del tiempo, el coste es claramente variable. Tiene unos costes fijos, que son el mínimo necesario para mantener el sistema activo. Lo bueno es que estos costes (cuando el sistema está en un uso muy bajo), suelen ser bastante reducidos, por lo cual se permite un coste general menor. Sólo cuando el sistema crece, aumenta el coste, porque el número de los nodos participantes aumentan, aunque este crecimiento suele ser temporal.

# 4. Tipos de alojamiento

Hay varios tipos de sistemas que se pueden tener en cuenta a la hora de montar un servicio orientado a web. Éstos se diferencian en dos grupos: uno con máquinas propias, y otro con máquinas en alquiler.

En el caso de las máquinas propias, los costes de implementación son caros, puesto que generalmente el coste de servidores es muy elevado.

Por otra parte, se puede optar por alquilar las máquinas, lo que se conoce como "*cloud*". Gracias a esto podemos tener servidores bajo demanda, temporalmente, sin hacer gastos iniciales y pagando sólo por el uso. Dentro de este grupo, se tienen varios tipos de servidores:

- ***Shared hosting***: Consiste en un servidor controlado por la empresa que proporciona el servicio, del cual se permite el uso parcial a cada cliente. El problema es que esta máquina es limitada, y si se da el caso de que todos los clientes que usan la misma máquina requieren carga de trabajo en el mismo momento, el servicio se puede ver afectado. Se suele gestionar a través de un panel, pero no permite seleccionar el software que se utiliza.
- **VPS (Virtual Private Server)**: Consiste en una máquina virtualizada, generalmente sobre un clúster, exclusiva para un cliente. Se puede elegir el sistema operativo a usar y se puede gestionar completamente. El rendimiento es limitado, puesto que está virtualizada.
- **Servidor dedicado**: Es un servidor equiparable a los que se usan en *housing*, con la diferencia que pertenece a la empresa que presta el servicio y se alquilan por meses. Son las que más rendimiento suelen dar, pero es importante tener en cuenta que pueden verse afectadas por un fallo de hardware y dejar de funcionar temporalmente o bien perder los datos contenidos en ellas.

Para crear sistemas en alta disponibilidad, se suelen usar servidores en *cloud*, ya que permiten pagar por el uso. Son estos sistemas los que se tendrán en cuenta para el trabajo.

# 5. ¿Cuándo se debe usar un sistema en alta disponibilidad?

En el ámbito empresarial es necesario hacer un estudio previo a la implantación de un sistema de alta disponibilidad nuevo, o cuando se piense en migrar uno convencional. Es importante tener en cuenta los siguientes elementos:

- Tráfico estimado semanal/mensual.
- Comportamiento de los usuarios: accesos constantes en el tiempo o en picos.

- Tipo de servicio: interno o público.
- Necesidad de que el sistema esté activo constantemente.
- Coste estimado por hora si se produce un fallo en el servicio.
- Necesidades propias del software.

No hay una "receta" a la hora de decidir si es necesario usar un sistema de alta disponibilidad. Normalmente, si el servicio tiene que estar en constante funcionamiento, puede ser un elemento decisivo para escoger este tipo de montajes. Aunque no sea así, es posible que se deba tener en cuenta este sistema si el número de accesos es en picos, no tanto por favorecer que no falle, sino para ahorrar costes cuando se tenga poca carga.

En general si el sistema es público, con un número alto de usuarios, suele ser recomendable usar este tipo de sistemas en alta disponibilidad, puesto que generalmente el coste es más ajustado y menor que el producido en un fallo de disponibilidad.

# Parte IV
# *Software*

Esta sección muestra ejemplos de software que se pueden utilizar para gestionar el tráfico en un sistema web (básico o complejo). No son necesariamente piezas que haya que utilizar simultáneamente, sino que podemos adaptarlas a nuestras necesidades escogiendo sólo algunas. Todo el software aquí ejemplificado es gratuito y libre.

## 6. Balanceador de carga

Un balanceador de carga es un *software* encargado de recibir el tráfico entrante a un sistema web y redistribuirlo a los *backends* pertinentes. Permite discriminar por tipo de petición, asignar pesos a cada *backend*, fiabilidad, seguridad, etc.

Además, un punto a destacar de este tipo de software, es que puede comprobar el estado de los *backends*, de modo que si uno de ellos falla, lo retira del balanceo y previene que se pierdan visitas.

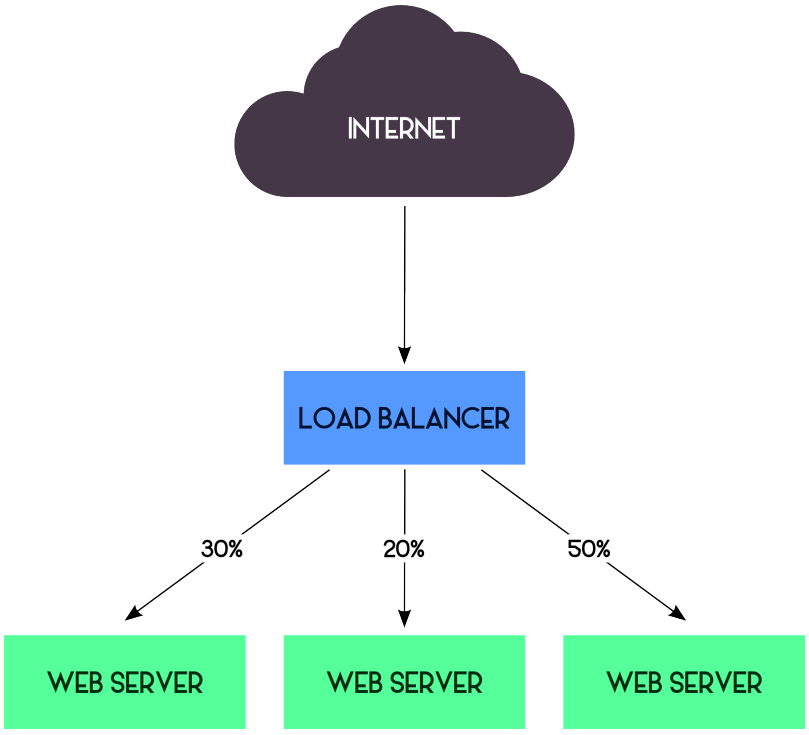


Figura 1: Esquema de un balanceador de carga.

Otro punto que hace muy interesante a los balanceadores de carga, es que pueden abstraer totalmente al usuario del montaje real que se ha utilizado, protegiéndolo de ataques directos a cada servidor.

# 7. Caché

Un servidor de caché es un *software* que de por sí, no contiene ningún dato del contenido a entregar, pero que sabe dónde está esa información, la solicita y tiene la capacidad de almacenarla para futuras peticiones. Permite contener tanto el tráfico como el coste de proceso en los *backends*, puesto que al tener ya la información puede entregarla al usuario.

Es importante tener en cuenta que se presupone que a una petición `x` siempre se obtendrá el resultado `y`. De no ser así, la caché no se podrá utilizar puesto que se podrían producir fallos entregando un contenido que no se corresponde con una petición dada. Por esto mismo, si se manejan sesiones, no se podrá cachear toda la información, puesto que cada usuario no tiene por qué recibir a misma información que otro usuario a una misma petición.

Para el caso que mejor funcionan los sistemas de caché es, por lo mencionado anteriormente, contenido estático: fotos, vídeos, fuentes, etc.

## 7.1. Funcionamiento de peticiones a una caché

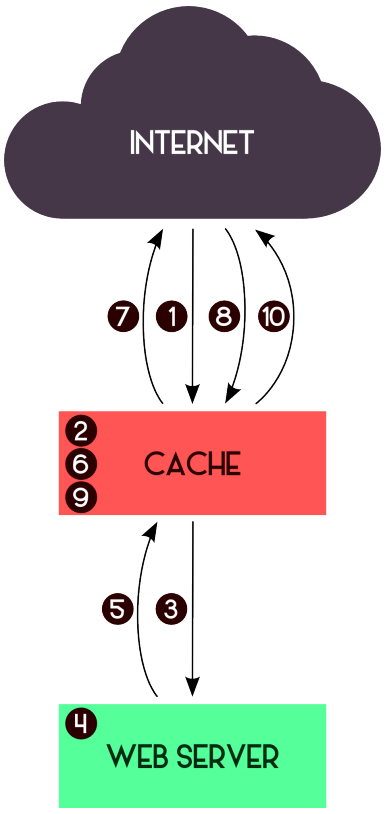


Figura 2: Esquema de funcionamiento de una caché.

1. Un cliente hace una petición `x`.
2. La caché comprueba si para esa petición tiene el resultado asociado.
3. Al no disponer de la información, hace la petición al *backend*.
4. El servidor web tramita la petición.
5. El servidor web devuelve la información `y` solicitada.
6. La caché almacena la información `y`.
7. La caché devuelve la información `y` al usuario.
8. Un cliente vuelve a hacer una petición `x`.
9. La caché comprueba si para esa petición tiene el resultado asociado.
10. Al disponer de la información `y`, la devuelve al usuario.

# 8. Servidor web

Un servidor web es un software capaz de recibir una petición HTTP, interpretarla y devolver el contenido solicitado.

# 9. Sistema gestor de bases de datos

Un sistema gestor de bases de datos es un software que permite almacenar información de un modo ordenado, de forma que se pueda acceder a ella posteriormente.

Dependiendo del tipo del SGBD que se utilice, será más sencillo desplegar varios nodos o replicar la información entre ellos.

## 9.1. Bases de datos relacionales

En este tipo de bases de datos, la información está organizada en tablas que se conectan por medio de relaciones.

El problema que supone cuando se quiere crear un sistema tolerante a fallos y escalable, es que estas bases de datos son muy estrictas y es complejo duplicarlas, hacer copias de seguridad o escalarlas, sobre todo cuando la cantidad de información que contienen es muy alta.

## 9.2. Bases de datos no relacionales

Las bases de datos no relacionales (NoSQL) son aquellas que se distinguen por no seguir un esquema de tablas estructurado. Hay cuatro grandes tipos de bases de datos NoSQL:

- Clave-Valor
- Documentales
- Orientadas a grafos
- Orientadas a objetos

Este tipo de bases de datos se caracterizan por la facilidad de escalación, velocidad, robustez y adaptabilidad a los datos que almacenan.

# 10. Respaldo de datos

Un *software* para respaldo de datos (*backup*) es aquel que realiza una copia de la información contenida en el servidor dentro del mismo en un lugar seguro donde

no se vayan a modificar estos ficheros y/o en otro servidor.

Existen varios tipos de sistemas de *backup*:

- **Completo**: Es aquel que hace una copia exacta de todos los ficheros solicitados. Consume tanto espacio como el origen de los datos por cada copia que se haga.
- **Diferencial**: En este caso se hace una copia completa de los datos, pero en sucesivas copias, sólo se copian los ficheros que han sido modificados, reduciendo considerablemente el espacio necesario con respecto al *backup* completo. Si se cambiaran todos los ficheros antes de cada copia, sería igual al *backup* completo.
- **Incremental**: Este tipo hace una copia completa inicial de los datos, pero en sucesivas copias, sólo guarda los cambios producidos en los ficheros, reduciendo muy significativamente el espacio necesario tras varias copias.

Hay que tener en cuenta a la hora de elegir que sistema de *backup* utilizar tanto el tamaño de los datos como la cantidad de tráfico que puede generar su transmisión. Dependiendo de la importancia de la información se pueden escoger la periodicidad de las copias locales y externas, no siendo en ningún momento incompatibles entre sí.

# 11. DNS

DNS (sistema de nombres de dominio), es un protocolo que permite convertir URLs a IPs donde se encuentran los servicios asociados.

DNS puede usarse sólo para resolver los nombres de dominio, pero también para balanceo de máquinas, añadiendo varios registros iguales pero apuntando a diferentes máquinas. Hay servicios más avanzados que, además, comprueban el estado de las máquinas a las que apuntan, y si no están accesibles, las quitan del balanceo (*failover*).

# Parte V
# Métodos para conseguir alta disponibilidad y adaptabilidad al tráfico

La alta disponibilidad se define como el porcentaje del tiempo que un servicio es accesible por un usuario de un total determinado. Cuando hablamos de alta disponibilidad en un sistema web, lo que se pretende es que el sistema sea accesible por un usuario la mayor cantidad de tiempo posible, reduciendo por tanto al mínimo posible el tiempo de fallo o indisponibilidad.

Este problema se puede enfocar desde dos vertientes diferentes, bien desde una visión puramente física del sistema o desde una versión abstracta, a más alto nivel, que definimos como lógica.

## 12. Alta disponibilidad a nivel físico

La alta disponibilidad física se basa básicamente en la redundancia. Para tener la mayor fiabilidad posible, debemos doblar todos los componentes, de modo que si uno falla, la máquina no entrará en estado de fallo puesto que puede seguir funcionando con el elemento adicional, dando tiempo al reemplazo de la pieza estropeada.

Para conseguir esta duplicidad se debe invertir en equipos que la permitan. Además se deberán tener al menos dos vías de alimentación eléctrica independientes y al menos dos conexiones de Internet.

Para poder conseguir alta disponibilidad de esta forma, es necesaria una gran inversión inicial y mensual, puesto que este tipo de equipos son mucho más caros y el alquiler o acceso a energía eléctrica e Internet redundados supone tener los servidores en algún alojamiento de *housing*.

## 13. Alta disponibilidad a nivel lógico

La alta disponibilidad a nivel lógico es parecida a la del nivel físico. Consiste también en tener duplicidad, pero en muchas ocasiones esa duplicidad es teórica o potencial, es decir, que no es necesario que sea real sino que el sistema esté preparado para comprobar su estado y corregirse antes del fallo o muy poco después.

En este caso, proponer duplicidades suele conllevar mucho menor coste, puesto que no es necesario invertir en elementos físicos, sino que se pueden duplicar las máquinas virtualizadas, que normalmente son mucho más baratas.

# 14. Adaptabilidad al tráfico

La adaptabilidad al tráfico supone que un sistema es capaz de controlar la cantidad de carga por la que está atravesando, y en base a ciertas métricas definidas, aumentar o reducir su capacidad (añadiendo y quitando nodos o redistribuyendo el tráfico).

Siguiendo esto, y al ser el sistema mucho más adecuado para el estado real en el que se encuentra, se pueden ajustar mucho los costes a la necesidad puntual. Se eliminan los sobrecostes de sobredimensionamiento que se dan en los sitemas típicos.

# 15. Replicación de bases de datos

## 15.1. Bases de datos relacionales

Para poder tener varias bases de datos relacionales con la misma información, es necesario configurarlas para que se comuniquen entre ellas avisándose de los cambios, y que de este modo mantengan coherencia.

En su forma más básica, hay dos métodos para poder conseguir este objetivo: replicación *master-slave* y replicación *master-master*.

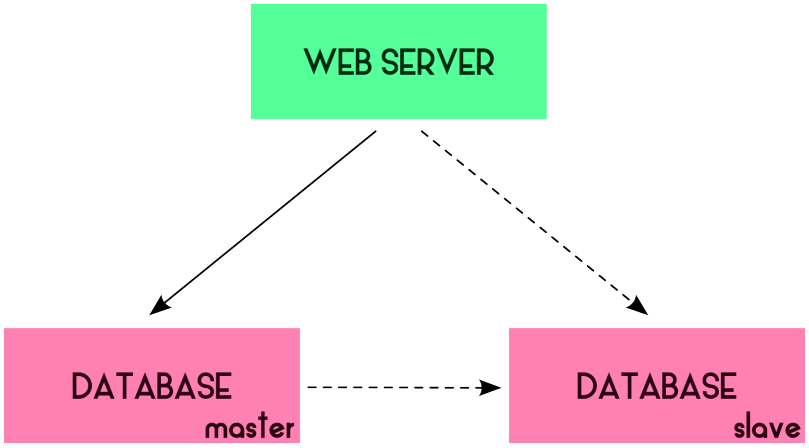


Figura 3: Esquema de replicación *master-slave*.

En el caso de utilizar una replicación *master-slave*, se necesitan al menos dos SGBD. Uno de ellos trabajará como *master*, esto quiere decir que deberá ser en

el que se realicen las actualizaciones de los datos. Este servidor irá almacenando los cambios que se vayan haciendo a los datos y se los entregará al *slave*. El *slave* irá leyendo todas las modificaciones y las irá realizando a sus datos, de modo que los datos sean exactamente los mismos en ambas bases de datos.

Será necesario preparar el *software* para que tenga en cuenta estas limitaciones. Es posible o bien usar sólo el nodo *master* y cambiar al *slave* en caso de fallo, o bien usar ambos simultáneamente para lectura de datos y sólo el *master* para inserción, modificación o borrado.

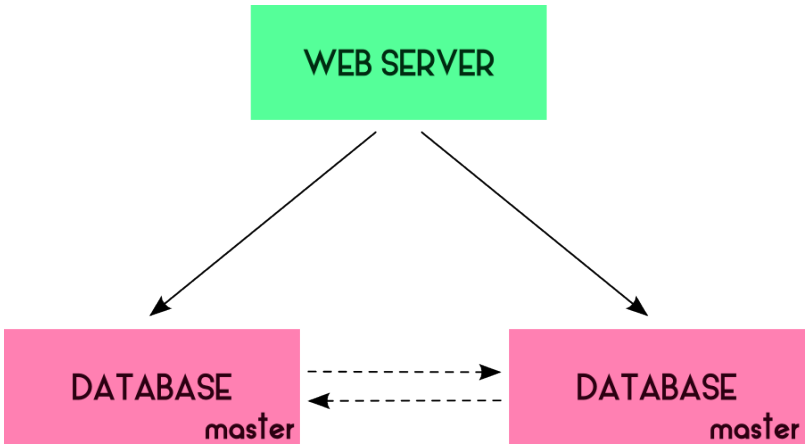


Figura 4: Esquema de replicación *master-master*.

En el caso de usar replicación *master-master*, la complejidad anterior en el software que da uso al sistema es menor, puesto que se pueden hacer todas las operaciones indistintamente en ambos nodos.

El problema que se presenta, aunque es sencillo de arreglar, es que nunca se podrá añadir un dato con el mismo identificador en ambos nodos, por lo que se tendrán que usar identificadores con incrementos que no se solapen.

## 15.2. Bases de datos no relacionales

En el caso de bases de datos NoSQL, la replicación está normalmente implementada de base, por lo que muchas veces es transparente. La propia base de datos es la que se encarga de distribuir los datos en los nodos participantes, discriminándolos en base a su clave. Además, los datos son consistentes, por lo que si un nodo falla el sistema es capaz de seguir funcionando sin problemas.

# Parte VI
# Monitorización

La monitorización siempre es importante en cualquier sistema, puesto que es necesario que conozcamos el estado real e histórico del sistema para que podamos tomar las medidas oportunas en caso de un error pasado o presente, además de preveer un error futuro.

Para monitorizar las máquinas, se pueden tener en cuenta muchos factores, siendo típicos los siguientes:

- Carga
- Memoria
- Disco
  - Espacio libre
  - Operaciones de entrada-salida
- Tráfico
- Número de procesos
- Usuarios (de la máquina) conectados

Además de estos datos, también se pueden obtener los estados de las bases de datos, del servidor web, de los sitemas de caché, etc.

Con toda esta información es mucho más fácil gestionar el sistema, comprobar necesidades que desconocíamos o encontrar errores los cuales serían mucho más difícil de comprobar y se necesitaría mucho más tiempo.

## 16. Métodos de monitorización

Actualmente existen varios métodos para monitorizar las máquinas: desde hacerlo con *scripts* propios que leen el estado de la máquina, hasta hacerlo con servicios preconfigurados que sólo requieren instalar un *software* en los servidores y configurar todo lo que se desea saber desde un panel.

En el estado intermedio, tenemos soluciones gratuitas que permiten tener el control total sobre lo que se necesita monitorizar, con un sistema sencillo de implementar, y paneles agradables para revisar la información obtenida. Gracias a estas soluciones, no es necesario invertir en un sistema privado, y tampoco hacer todo el control desde cero.

# 17. Cacti: propuesta de *software* de monitorización

Un ejemplo de *software* gratuito, de fácil instalación y gestión es Cacti (`www.cacti.net`).

Para usarlo, es necesario diferenciar dos partes: el servidor de control, que se encarga de solicitar la información a las máquinas a monitorizar, guardarla y mostrarla; y las propias máquinas monitorizadas.

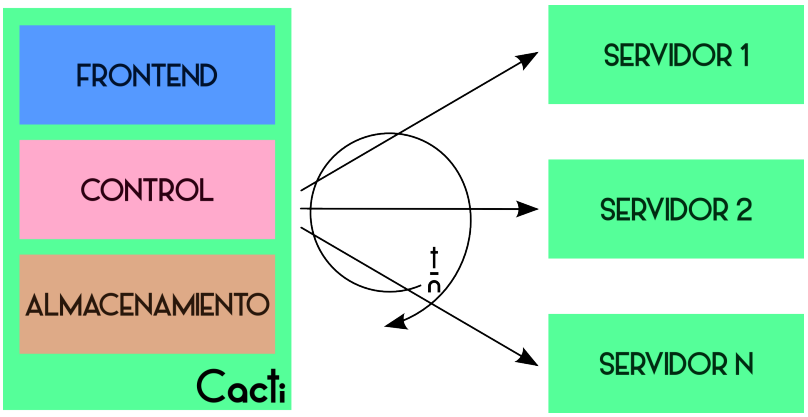


Figura 5: Esquema de funcionamiento de Cacti.

## 17.1. Obtención de la información

Para obtener la información de las máquinas, símplemente es necesario crear un *host* nuevo en el panel de Cacti, y seleccionar los datos que se necesitan leer. A partir de entonces, y con una frecuencia de tiempo determinada, el sistema se encargará de solicitar la información a los *hosts* y almacenarla para su posterior representación.

Para que el sistema monitorizado responda a las solicitudes de información, es necesario que cuente con `snmpd`, demonio del protocolo SNMP (*Simple Network Management Protocol*) que permite transmitir la información del sistema que necesitamos. Es necesario tener en cuenta que SNMP trabaja sobre UDP, por lo que la transmisión no es confiable y es posible que se pierdan o ignoren paquetes cuando la máquina a monitorizar esté saturada. Por esto es posible que haya periodos de tiempo en los que la información no sea completa.

## 17.2. Representación

Para representar la información, Cacti utiliza RRDtool (`http://oss.oetiker.ch/rrdtool/`), una herramienta que permite guardar y graficar información.

Gracias a esta herramienta, es muy sencillo comprobar los estados de las máquinas, compararlos, sumarlos, etc. puesto que se pueden crear gráficas nuevas que se nutran de diferentes fuentes de datos.

# Parte VII
# Guía orientativa para montaje de sistema de alta disponibilidad en servicios *cloud*

## 18. Consideraciones previas

Es importante tener en cuenta que la solución aquí planteada no es única, pero se presupone como una buena representación de un sistema perfectamente viable.

Esta propuesta se basará en que el sistema sea plenamente accesible, pero se intentará reducir al mínimo los gastos que supondría su implementación, consiguiendo de este modo un sistema robusto, capaz de gestionar el mayor número de peticiones posibles, con la menor inversión. Puesto que realizar un montaje extenso y costoso no tiene sentido para un sistema pequeño, la solución planteada lo será para el supuesto en el que el sistema en cuestión sea suficientemente grande como para ser necesario, con un número muy alto de usuarios y visitas por minuto.

Este sistema está planteado para la web ficticia `www.larevistaweb.com`, que cuenta con dos tipos de usuarios: administradores de la web, que tienen acceso a un panel de control, y lectores, que pueden ver los contenidos publicados y comentar sobre los mismos. La web estará creada teniendo en cuenta el gran número de accesos, por lo que estará enfocada a ofrecer contenido estático cuando sea posible, separando en diferentes subdominios el contenido:

- **www.larevistaweb.com**: Contenidos públicos, estáticos, cambian cada relativamente poco tiempo.
- **cdn.larevistaweb.com**: Contenidos estáticos de medios, tales como imágenes, vídeos y audio.
- **adm.larevistaweb.com**: Panel de administración, acceso restringido.
- **tlk.larevistaweb.com**: Control de comentarios, contenido dinámico y accesible a usuarios registrados.

Con referencia al *software*, se utilizarán los siguientes:

- **DNS**: No se usará ninguno propio, sino el sistema de DNS ofrecido por el vendedor del dominio.
- **Balanceador de carga**: HAProxy 1.5.8

- **Caché**: Varnish 3.0.2
- **Servidor web**: Nginx 1.2.1
- **SGBD**: MySQL 5.5
- **Respaldo de datos**: Backupninja 1.0.1

Todas las máquinas ejecutarán Debian 7.0 de 64 bits. Las máquinas serán virtualizadas en *cloud*, por motivos de precio, velocidad y adaptabilidad.

# 19. Sistema propuesto

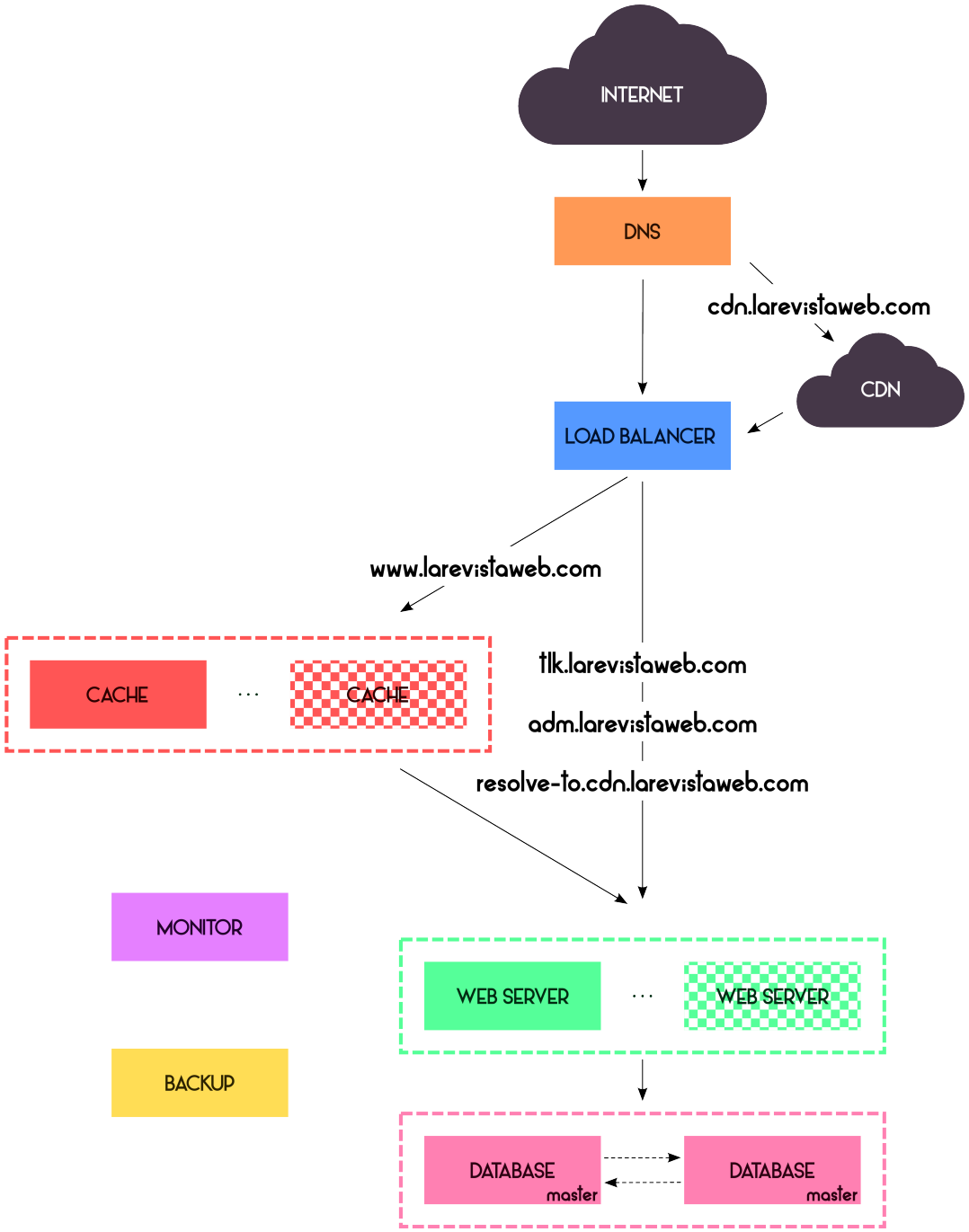


Figura 6: Esquema del sistema propuesto.

La CDN (*Content Delivery Network*) usada anteriormente puede tener un esquema parecido al de la figura 7. Se diferencia con el sistema de caché en que normalmente almacena los datos por mucho más tiempo, suelen ser ficheros de mayor tamaño y las máquinas están distribuidas espacialmente.

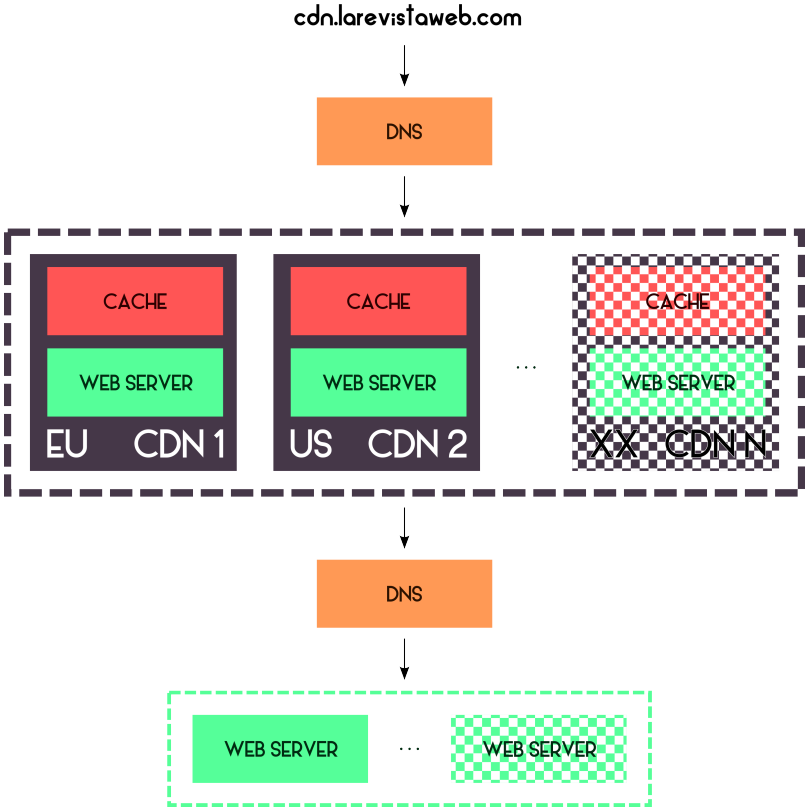


Figura 7: Esquema de la CDN.

## 19.1. Explicación del sistema propuesto

Para acceder al sistema y poder resolver el nombre de dominio, se hará uso de unas DNS genéricas, siendo válidas las que suele proporcionar cualquier vendedor de dominios.

El balanceador de carga será un HAProxy, tal y como se ha comentado previamente. En principio sólo aparece un único balanceador, esto es porque aunque haya un alto número de peticiones, éstas se manejan muy rápido y por lo general una única máquina es capaz de soportar todo este tráfico. En el caso de que se haya preparado el sistema y se observe a través de las gráficas de monitorización que no es suficiente la capacidad, se pueden añadir tantos balanceadores como sean necesarios. Para ello, basta con copiar el fichero de configuración del HAProxy a la máquina o máquinas nuevas y añadir los nuevos registros pertinentes en las DNS.

El balanceador, dependiendo del dominio de entrada, distribuirá la petición al sistema pertinente:

- **www.larevistaweb.com**: Puesto que las peticiones que entren a esta URL serán de lectura de la web, el contenido resultante será código HTML que será de por sí cacheable. Es por esto que la petición se envía a los servidores

de caché, que a su vez harán la petición correspondiente a los servidores web, y almacenarán el resultado, guardándolo para futuras peticiones idénticas.

- **adm.larevistaweb.com** y **tlk.larevistaweb.com**: Puesto que estas URL hacen referencia al *backend* y al servicio de comentarios, que no son cacheables por tener al usuario conectado entre otros motivos, la solicitud se hará directamente a los servidores web, saltándose la caché y CDN.
- **cdn.larevistaweb.com**: En esta URL están referenciados todos los ficheros estáticos de la web (por código), tales como imágenes, vídeos, audios, etc.

Con respecto a la caché, sus máquinas guardarán el resultado de las peticiones realizadas un tiempo determinado. Este tiempo debe ser ajustado basándose en diferentes incógnitas que provocarán un funcionamiento diferente, tales como tiempo medio de cambio de la web, número de peticiones por segundo, etc.

Después de las cachés, la petición se entregará a los servidores web, que gestionarán la petición, accederán a las bases de datos pertinentes, y entregarán el resultado solicitado.

El funcionamiento de la CDN es parecido al de la caché, puesto que no deja de ser lo mismo. La diferencia radica en que los servidores de CDN suelen estar espaciados en diferentes países o continentes, de modo que se pueda distribuir el tráfico en base al servidor más cercano al usuario. Al hacer esto, y más con ficheros grandes, se pueden reducir tanto la latencia como el tiempo de transferencia de los objetos.

El método de resolución de la CDN es mediante DNS, tanto para la entrada como para la salida, siguiendo el siguiente esquema:

```
A      @              2.2.2.1
A      cdn            9.9.9.1
A      cdn            9.9.9.2
A      cdn            9.9.9.3
CNAME  resolve-to.cdn @
```

El motivo para utilizar la CDN mediante DNS y no con el balanceador de carga, es porque se puede reutilizar para más webs, y por tanto es mejor separarlo del sistema principal.

Las bases de datos son relacionales y siguen un esquema *master-master*. En principio no se ha contemplado añadir más unidades (se podría hacer un esquema en anillo), pero como se utilizarán cachés y no hay excesivos usuarios que harán uso interno del sistema (que conlleva más entrada/salida a la base de datos), es previsible que sean suficientes.

Se ha elegido el esquema *master-master* puesto que si uno de los nodos cae, es más sencillo mantener el sistema funcionando y recuperarlo una vez se ha arreglado el fallo.

# 20. Configuraciones de servicios

## 20.1. DNS

Como se ha comentado anteriormente, un sistema genérico de DNS es suficiente para controlar este sistema, puesto que sólo se usará para apuntar todo el dominio al balanceador y el subdominio cdn a la CDN. Como se ha escrito anteriormente, el dominio y los subdominios `www`, `tlk`, `adm` y `resolve-to.cdn` apuntarán al balanceador de carga. El subdominio `cdn` apuntará a todas las máquinas de la CDN.

Para añadir robustez al sistema, se puede utilizar un servicio de DNS con IP *failover*, que se encarga de comprobar el buen estado de los destinos de las DNS, y en caso de fallo cambia la IP por una de respaldo.

Este mismo sistema lo podemos usar para añadir robustez en caso de que la máquina balanceadora entrara en estado de fallo, poniendo directamente la IP a los servidores web, haciendo que la web siga operativa hasta reestablecer la máquina de balanceo.

## 20.2. Balanceador

Para preparar el balanceador de carga, debemos instalar HAProxy en el servidor. Puesto que no está en el repositorio genérico, deberemos activar los `backports`. Para ello debemos editar el archivo `/etc/apt/sources.list` añadiendo:

```
deb http://cdn.debian.net/debian wheezy-backports main
```

Una vez hemos añadido el repositorio, actualizamos la lista de paquetes e instalamos el programa:

```
# apt-get update
# apt-get install haproxy
```

Ahora, cambiaremos el fichero de configuración para indicarle cómo tratar las peticiones y dónde enviarlas:

*/etc/haproxy/haproxy.cfg*

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
```

```
        stats socket /run/haproxy/admin.sock mode 666
        stats timeout 30s
        maxconn 131070
        ulimit-n 262500
        nbproc 1
        user haproxy
        group haproxy
        daemon

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        retries 3
        option redispatch
        maxconn 65535
        stats enable
        stats auth admin:statUS
        timeout connect 5000
        timeout client  50000
        timeout server  50000

backend cache
        server c1  3.3.3.1:80  check
        server c2  3.3.3.2:80  check

backend webserver
        server w1  4.4.4.1:80  check
        server w2  4.4.4.2:80  check

listen web 2.2.2.1:80
        mode http
        option httpchk GET /alive.html
        balance leastconn
        option forwardfor
        acl use_cache hdr_reg(host) ^www.*
        use_backend cache if use_cache
        default_backend webserver
        option httpclose
```

Una vez cambiado el fichero de configuración, reiniciamos el servicio:

```
# service haproxy restart
```

A partir de ahora, el balanceador repartirá el tráfico. Podremos consultar su estado y el de los *backends* accediendo a `http://2.2.2.1/haproxy?stats` con el usuario y contraseña especificados, en nuestro caso `admin:statUS`.

Si queremos añadir más máquinas al balanceo, tanto de las cachés como de los servidores web, es tan sencillo como añadir la línea correspondiente con el nombre y la nueva ip al `backend` que corresponde y reiniciar el servicio.

## 20.3. Caché

Primero es neceasario instalar `varnish` en la máquina, esto se hará usando `apt-get`:

```
# apt-get install varnish
```

Es importante comprobar en `/etc/default/varnish` que está activada la opción de arranque al inicio, para ello hay que comprobar la opción `START` teniendo que tener `yes` como valor.

El archivo de configuración por defecto es `/etc/varnish/default.vcl`. Este fichero se compone de varias funciones diferentes:

- `vcl_recv`: función de entrada, se encarga de gestionar a qué proceso entregar la petición dependiendo de lo configurado. Es en este punto donde se añade la gran parte de lógica que queremos que haga, como por ejemplo definir cuáles son a los *backends* a los que se debe entregar la petición dependiendo de las distintas variables que queramos considerar.
- `vcl_pipe`: función para crear una tubería TCP y no tener en consideración el contenido de las peticiones HTTP. Se utiliza cuando hay algún valor extraño que no se puede gestionar y se entrega al servidor web para que sea él quien lo gestione.
- `vcl_pass`: función que provoca que no se compruebe si el objeto está en caché, directamente se pide al *backend*.
- `vcl_hash`: función que crea un hash del objeto para poder comprobar si el objeto se encuentra en la caché, ya que sigue un sistema de clave-valor.
- `vcl_hit`: función a la que se le invoca cuando el resultado de comprobar en la caché si el objeto existe, es positivo.
- `vcl_miss`: función a la que se le invoca cuando el resultado de comprobar en la caché si el objeto existe, es negativo.
- `vcl_fetch`: función que solicita el objeto pedido al *backend*.
- `vcl_deliver`: función que entrega el contenido al usuario.

- `vcl_error`: función que gestiona los errores que se hayan podido producir en los procesos.
- `vcl_init`: función a la que se llama cuando se carga el fichero de configuración.
- `vcl_fini`: función a la que se llama cuando se ha acabado de leer el fichero de configuración.

Para este caso sólo sería necesario crear los dos *backends* (fuera de las funciones) y las condiciones de caché. De este modo tendríamos el fichero `default.vlc` con el siguiente contenido:

*/etc/varnish/default.vcl*

```
backend  ws1{
    .host = "4.4.4.1";
    .port = "80";
    .probe = {
         .url = "/alive.html";
         .interval = 5s;
         .timeout = 1 s;
         .window = 5;
         .threshold = 3;
    }
}

backend  ws2{
    .host = "4.4.4.2";
    .port = "80";
    .probe = {
         .url = "/alive.html";
         .interval = 5s;
         .timeout = 1 s;
         .window = 5;
         .threshold = 3;
    }
}

director webservers round-robin {
    {
        .backend = ws1;
    }
    {
        .backend = ws2;
    }
```

```
}

sub vcl_recv {
    set req.backend = webservers;
    if (req.restarts == 0) {
        if (req.http.x-forwarded-for) {
            set req.http.X-Forwarded-For =
                req.http.X-Forwarded-For + ", " + client.ip;
        } else {
            set req.http.X-Forwarded-For = client.ip;
        }
    }
    if (req.request != "GET" &&
        req.request != "HEAD" &&
        req.request != "PUT" &&
        req.request != "POST" &&
        req.request != "TRACE" &&
        req.request != "OPTIONS" &&
        req.request != "DELETE") {
        return (pipe);
    }
    if (req.http.Authorization || req.http.Cookie) {
        return (pass);
    }
    return (lookup);
}
```

En este caso definimos los *backends* con la directiva `backend` con el destino y la comprobación de que realmente están funcionando (`.probe`), leyendo el fichero `alive.html`. Una vez añadidos los *backends* los incorporamos en un grupo (`director`). Con el grupo, indicamos en el apartado de recepción que el backend que se va a usar global es `webservers`.

Las siguientes sentencias de control son para que si entra una petición desconocida se entregue directamente a los *backends* y que si el usuario está conectado no se cachee.

Después de configurar el servicio, es necesario reiniciarlo:

```
# service varnish restart
```

## 20.4. Servidor web

Para preparar el servidor web, es necesario instalar `nginx`, además de `php5-fpm` (puesto que la web planteada está desarrollada en este lenguaje) y `mysql-client`:

```
# apt-get install nginx php5-fpm mysql-client
```

Los ficheros de configuración de Nginx se encuentran en la ruta `/etc/nginx`, conteniendo los siguientes ficheros y directorios (están listados los más importantes):

- `/etc/nginx/sites-available/`: en este directorio están los *virtualhosts* preparados pero que no están usándose necesariamente.
- `/etc/nginx/sites-enabled/`: en este directorio se encuentran enlaces a la carpeta `sites-available` para activar los *virtualhosts*.
- `/etc/nginx/nginx.conf`: esta es la configuración por defecto de Nginx y de la que cuelgan el resto.
- `/etc/nginx/conf.d/`: en este directorio es donde se ponen las configuraciones personalizadas que se cargarán en todos los *virtualhosts*.

Primero debemos configurar los *virtualhosts*, que en este caso serán tres, uno para cada servicio:

*/etc/nginx/sites-available/larevistaweb_com.conf*

```
server {
        root /var/www/larevistaweb.com;
        index index.php;

        server_name larevistaweb.com www.larevistaweb.com
                            resolve-to.cdn.larevistaweb.com;

        location / {
                try_files $uri $uri/ /index.php?args;
        }

        location ~ \.php$ {
                fastcgi_pass unix:/var/run/php5-fpm.sock;
                fastcgi_index index.php;
                include fastcgi_params;
        }

        location ~* \.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$ {
                expires max;
                log_not_found off;
        }

        access_log  /var/log/nginx/larevistaweb.com_access.log;
        error_log   /var/log/nginx/larevistaweb.com_error.log;
}
```

*/etc/nginx/sites-available/tlk_larevistaweb_com.conf*

```
server {
        root /var/www/tlk.larevistaweb.com;
        index index.php;

        server_name tlk.larevistaweb.com;

        location / {
                try_files $uri $uri/ /index.php?args;
        }

        location ~ \.php$ {
                fastcgi_pass unix:/var/run/php5-fpm.sock;
                fastcgi_index index.php;
                include fastcgi_params;
        }

        location ~* \.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$ {
                expires max;
                log_not_found off;
        }

        access_log  /var/log/nginx/tlk.larevistaweb.com_access.log;
        error_log   /var/log/nginx/tlk.larevistaweb.com_error.log;
}
```

*/etc/nginx/sites-available/adm_larevistaweb_com.conf*

```
server {
        root /var/www/adm.larevistaweb.com;
        index index.php;

        server_name adm.larevistaweb.com;

        location / {
                try_files $uri $uri/ /index.php?args;
        }

        location ~ \.php$ {
                fastcgi_pass unix:/var/run/php5-fpm.sock;
                fastcgi_index index.php;
                include fastcgi_params;
        }

        location ~* \.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$ {
```

```
                expires max;
        }

        access_log  /var/log/nginx/adm.larevistaweb.com_access.log;
        error_log   /var/log/nginx/adm.larevistaweb.com_error.log;
}
```

Como se puede comprobar, los tres ficheros son similares, puesto que en principio la configuración de los tres es similar (no requieren de elementos específicos).

Dentro de los ficheros de configuración se pueden observar las diferentes directivas:

- `root`: carpeta en la que está contenida la web.
- `index`: ficheros que se leerán por defecto si no se especifica ninguno.
- `server_name`: nombre de dominio por el cual saber si se debe mostrar un *virtualhost* u otro.
- `location`: las cabeceras *location* permiten hacer configuraciones específicas dependiendo de la carpeta o ficheros determinados.

  `fastcgi_pass`: lugar en el que se encuentra el servidor PHP escuchando.

  `fastcgi_index`: nombre del fichero a leer por defecto.

  `include`: incluye el fichero especificado, en este caso el fichero que contiene las especificaciones genéricas del servidor PHP.

  `expires`: añade una cabecera que indica a la caché que puede haber por delante el tiempo que, en principio, debe cachear.
- `access_log`: fichero en el que se guardará un registro de las solicitudes.
- `error_log`: fichero en el que se guardará un registro de los errores de servidor.

Es importante tener en cuenta que el registro de accesos y errores provoca lentitud cuando el número de peticiones es muy alta, por lo que puede ser recomendable no guardar esta información y sólo activarla en el caso de que se encontrase algún problema.

En el caso de que se requiriese utilizar HTTPS en alguna web, sería necesario añadir los siguientes elementos dentro de `server`:

```
listen 443;
ssl on;
ssl_certificate cert.pem;
ssl_certificate_key cert.key;
ssl_session_timeout 5m;
```

```
ssl_protocols SSLv3 TLSv1;
ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv3:+EXP;
ssl_prefer_server_ciphers on;
```

De este modo, modificamos el servidor para que escuche a través del puerto genérico de HTTPS, activamos el `ssl` e indicamos cuales son las credenciales y los métodos de cifrado.

Una vez configurado el servidor, es necesario reiniciarlo:

```
# service nginx restart
```

## 20.5. Bases de datos

Para este apartado es necesario diferenciar los dos participantes, que serán *master-master*, cada uno con una IP diferente:

1. **BD1**: 8.8.8.1
2. **BD2**: 8.8.8.2

Para preparar ambos servidores, hay que instalar el *software* necesario:

```
# apt-get install mysql-server mysql-client
```

Para este ejemplo, pondremos como contraseña de `root` en ambas máquinas `rootpass`.

Una vez instalados los paquetes necesarios, hay que editar el fichero de configuración para así permitir la replicación entre los dos nodos, quedando como sigue:

*/etc/mysql/my.cnf*

```
[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock

[mysqld_safe]
socket          = /var/run/mysqld/mysqld.sock
nice            = 0

[mysqld]
user                    = mysql
pid-file                = /var/run/mysqld/mysqld.pid
socket                  = /var/run/mysqld/mysqld.sock
port                    = 3306
basedir                 = /usr
```

```
datadir                 = /var/lib/mysql
tmpdir                  = /tmp
lc-messages-dir         = /usr/share/mysql
skip-external-locking
key_buffer              = 16M
max_allowed_packet      = 16M
thread_stack            = 192K
thread_cache_size       = 8
myisam-recover          = BACKUP
max_connections         = 1000
query_cache_limit       = 1M
query_cache_size        = 16M
server-id               = 1
log_bin                 = /var/log/mysql/mysql-bin.log
expire_logs_days        = 10
max_binlog_size         = 100M
binlog-ignore-db        = performance_schema
binlog-ignore-db        = information_schema
binlog-ignore-db        = mysql
log-slave-updates
replicate-ignore-db     = performance_schema
replicate-ignore-db     = information_schema
replicate-ignore-db     = mysql
relay-log               = mysqld-relay-bin

[mysqldump]
quick
quote-names
max_allowed_packet      = 16M

[mysql]

[isamchk]
key_buffer              = 16M

!includedir /etc/mysql/conf.d/
```

Es importante asignar un *id* diferente a cada nodo, para ello hay que cambiar el valor en `server-id`.

Hay varias formas de configurar sobre qué bases de datos hacer la replicación. En este caso haremos replicado de todas las bases de datos exceptuando las propias de MySQL que deben ser únicas para cada servidor.

Una vez configurados los MySQL, debemos crear los usuarios que llevarán a cabo la replicación. Para ello hay que acceder a MySQL:

```
# mysql -uroot -prootpass
```

Una vez dentro, debemos ejecutar lo siguiente, que creará el usuario `replicador` con la contraseña `replicante` que tendrá acceso total a todas las bases de datos:

```
create user 'replicador'@'%' identified by 'replicante';
grant replication slave on *.* to 'replicador'@'%';
```

Después de configurar los *master* hay que configurar los *slave* (para poder replicar de la segunda máquina, ambos tienen que ser *master* y *slave* simultáneamente). Para ello, primero hay que conocer el estado de los *master*:

```
show master status;
```

Ahora deberemos ejecutar lo siguiente en cada uno de los MySQL, especificando la IP correspondiente al otro servidor (`MASTER_HOST`), el fichero de replicación (`MASTER_LOG_FILE`) y la posición (`MASTER_LOG_POS`):

```
slave stop;
CHANGE MASTER TO MASTER_HOST = '8.8.8.2', MASTER_USER = 'replicador',
MASTER_PASSWORD = 'replicante', MASTER_LOG_FILE = 'mysql-bin.000001',
MASTER_LOG_POS = 346;
slave start;
```

Para comprobar el estado del *slave* podemos ejecutar:

```
show slave status;
```

Así podremos ver si está funcionando correctamente y si ha replicado al *master*. Una vez acabada esta configuración, el estado `Slave_IO_State` deberá ser "`Waiting for master to send event`" y `Seconds_Behind_Master` igual a `0`.

Además, para prevenir la posible concurrencia si se insertasen dos valores simultáneamente, hay que modificar las variables de `AUTO_INCREMENT`, para que una máquina tome valores pares y otra impares, poniendo un *offset* en una de `1` y en la otra de `2`. Para ello basta ejecutar:

```
SET @@auto_increment_increment=2;
SET @@auto_increment_offset=1;
```

A partir de ahora, todo lo que hagamos en cada una de las bases de datos se replicará a la otra y viceversa.

## 20.6. CDN

Las máquinas de la CDN contienen un Varnish y un Nginx. Se deberán instalar como se ha indicado anteriormente. Los ficheros de configuración son los siguientes en este caso:

*/etc/varnish/default.vcl*

```
backend default {
        .host = "127.0.0.1";
        .port = "8080";
        .connect_timeout = 1s;
        .first_byte_timeout = 5s;
        .between_bytes_timeout = 2s;
}

sub normalize_user_agent {
        set req.http.X-UA = "cdn";
}

sub vcl_recv {
        call normalize_user_agent;
        set req.url = regsub(req.url, "\&.*", "");
        remove req.http.cookie;
        if (req.http.x-forwarded-for) {
                set req.http.X-Forwarded-For = req.http.X-Forwarded-For
                                               + ", " + client.ip;
        } else {
                set req.http.X-Forwarded-For = client.ip;
        }
        if (req.backend.healthy) {
                set req.grace = 1m;
        } else {
                set req.grace = 1h;
        }
        if (req.url ~ "\.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$") {
                return (lookup);
        }
        return (pass);
}

sub vcl_fetch {
        set beresp.http.X-UA = req.http.X-UA;
        set beresp.http.Vary = "X-UA";
        set beresp.grace = 1h;
        unset beresp.http.set-cookie;
```

```
        if (beresp.ttl < 1h) {
        #std.log("Adjusting TTL");
                set beresp.ttl = 1h;
        }
        if (req.url ~ "\.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$") {
                unset beresp.http.set-cookie;
        }
        return (deliver);
}
```

*/etc/nginx/sites-available/cdn.conf*

```
server {
        listen 8080 default;
        server_name _;
        server_name_in_redirect off;
        location ~* \.(js|css|png|jpg|jpeg|gif|ico|mp3|mp4|flv|ogg)$ {
                resolver 8.8.8.8;
                proxy_set_header        X-Real-IP $remote_addr;
                proxy_set_header        X-Forwarded-For
                                        $proxy_add_x_forwarded_for;
                proxy_pass http://resolve-to.$host$uri;
                proxy_store /var/www/cache/$host$uri;
                proxy_store_access user:rw group:rw all:r;
                proxy_cache_valid 200 302 24h;
                proxy_cache_valid 404 1m;
                proxy_cache_use_stale updating;
        }
        access_log off;
        error_log /dev/null crit;
}
```

En este caso, usaremos Varnish y Nginx como cachés. El funcionamiento es el siguiente:

1. Varnish recibe la petición.
2. Elimina el *user_agent* para que se cacheen todos los objetos por igual sin importar el usuario.
3. Elimina todas las *cookies*.
4. Añade en la cabecera `X-Forwarded-For` su IP.
5. Envía la petición al Nginx.
6. Guarda la información en `/var/www/cache/dominio/URI`.

7. Pide la información a `http://resolve-to.dominio/URI`.

La configuración en todas las máquinas pertenecientes a la CDN es equivalente, transparente a los dominios y los elementos solicitados. Gracias a esto, si se quisiera usar para otros dominios, no sería necesario cambiar nada de la configuración.

## 20.7. Respaldo de datos

Para hacer copias de seguridad de los datos, usaremos backupninja, tal y como se ha comentado anteriormente. Los *backups* serán incrementales, por motivos de espacio.

Hay que distinguir dos actores: la máquina que va a hacer sus copias de seguridad, y la que almacenará las mismas. En el caso de la máquina que guardará las copias, sólo es necesario instalar lo siguiente:

```
# apt-get install rdiff-backup
```

Además, por motivos de seguridad, crearemos un usuario nuevo llamado `backups` y crearemos el directorio `/backup`, donde se realizarán las copias:

```
# adduser backups
# mkdir /backup
# chown backups. /backup
# chmod 700 /backup
```

En el caso del resto de las máquinas, además del paquete `rdiff-backup` necesitamos instalar `backupninja`:

```
# apt-get install rdiff-backup backupninja
```

Una vez tenemos todos los paquetes instalados, ya podemos configurar el `backupninja` para que haga backup de los ficheros que necesitemos. Por lo general, haremos *backup* de todos los ficheros de configuración, o de las web. Para configurar las copias, hay que ejecutar:

```
# ninjahelper
```

Una vez ejecutado, aparece un panel por el cual vamos configurando las acciones que queremos que lleve a cabo:

1. Pulsamos "new".
2. Pulsamos "rdiff".
3. Marcamos "src".
4. Añadimos todos los directorios o ficheros de los que queremos que se haga copia.

5. Añadimos todos los directorios o ficheros que no queremos copiar.
6. Pulsamos "dest".
7. En "keep" ponemos el número de días de los que queremos tener históricos.
8. En "dest_directory" indicamos la carpeta donde se guardarán todos los ficheros en destino.
9. En 'dest_host" decimos la IP de la máquina de *backup*.
10. En "dest_user" escribimos el usuario con el que se conectará a la máquina de destino.
11. En "dest_type" dejamos el valor por defecto `remote`.
12. Pulsamos "conn". La máquina creará un par de claves pública-privada y se conectará a la máquina de backups para copiarlas. De este modo, de ahora en adelante, podrá conectarse sin necesidad de introducir la contraseña.
13. Pulsamos "finish". A partir de ahora se creará una copia diaria de los datos.

El fichero de configuración (se puede editar directamente), quedaría como sigue:

*/etc/backup.d/90.rdiff*

```
[source]
type = local
keep = 60D

include = /var/spool/cron/crontabs
include = /var/backups
include = /etc
include = /root
include = /home
include = /usr/local/*bin
include = /var/lib/dpkg/status*
exclude = /home/*/.gnupg
exclude = /home/*/.local/share/Trash
exclude = /home/*/.Trash
exclude = /home/*/.thumbnails
exclude = /home/*/.beagle
exclude = /home/*/.aMule
exclude = /home/*/gtk-gnutella-downloads
exclude = /var/cache/backupninja/duplicity

[dest]
type = remote
```

```
directory = /backup/pruebastfg
host = 5.5.5.1
user = backups
```

En el caso de las bases de datos, necesitamos crear otra entrada, pero en este caso de tipo `mysql`. Haremos copia de todas las bases de datos, comprimidas y con el usuario `debian`. Las copias se harán en `/var/backups/mysql`, que al estar incluido en el *backup* anterior, también se copiará.

El fichero de configuración quedaría como sigue:

*/etc/backup.d/20.mysql*

```
hotcopy = no
sqldump = yes
compress = yes
backupdir = /var/backups/mysql
databases = all
configfile = /etc/mysql/debian.cnf
```

# Parte VIII
# Reflexión personal

## 21. Análisis teórico de beneficios

El análisis hecho en este apartado es completamente teórico, con números considerados para satisfacer supuestos específicos. Se tiene en cuenta lo siguiente:

- Incrementar la capacidad de los servidores propios implica un coste de 3.000 €.
- El beneficio obtenido en base a los usuarios es de un 12 %.
- El coste de los servidores en *cloud* es del 10 % del número de usuarios (puesto que se adecua a éstos) .

### 21.1. Sistema con servidores propios

Si se utilizan servidores propios, es necesario hacer una inversión grande en el caso de que el número de usuarios crezca, puesto que hay que adecuar la capacidad de trabajo al tráfico, para que se puedan satisfacer todas las peticiones y no se pierdan usuarios o el sistema entre en estado de fallo. En un caso positivista, esto no tiene por qué suponer un problema, puesto que el número de usuarios crece o se mantiene.

Este caso puede observarse en la figura 8, donde se muestra una relación entre gastos e ingresos, con un cómputo de beneficios mensual. Aunque algún mes haya pérdidas, se puede considerar como inversión. En general, el volumen de beneficios aumenta con el tiempo.

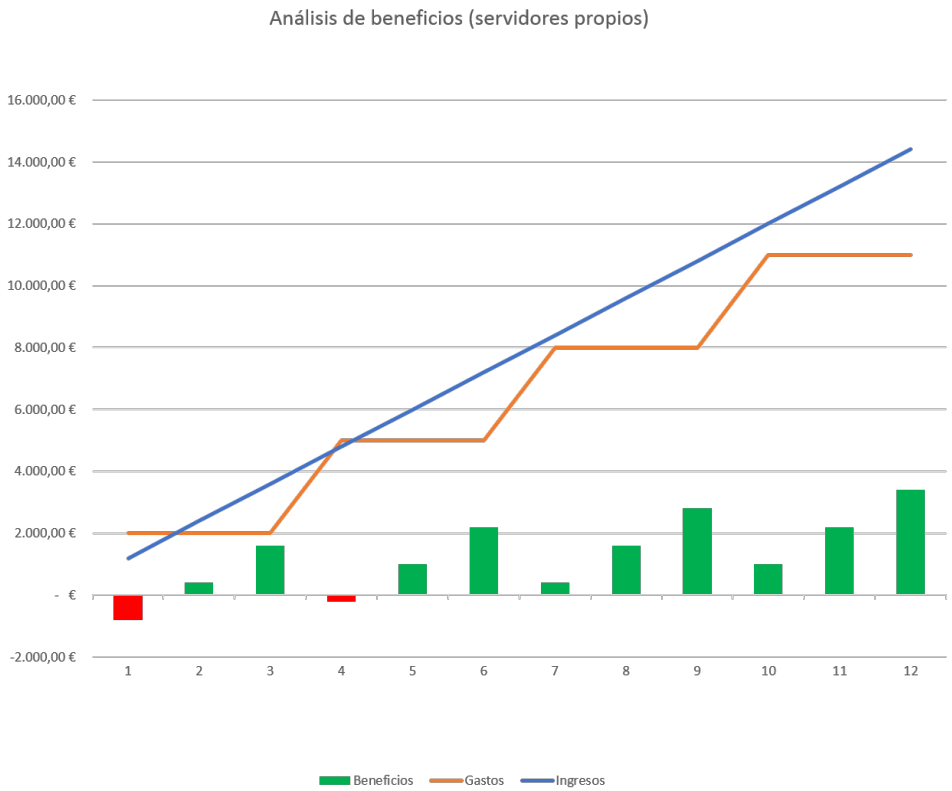


Figura 8: Gráfica de beneficios con servidores propios (optimista).

El problema que suponen estas inversiones, es que el gasto es permanente en el tiempo. Por tanto, si se necesita aumentar la capacidad del sistema, esta será permanente.

En un caso negativista, aunque posible, en el cual el número de usuarios (y por tanto ingresos) disminuya, el coste del sistema podrá llegar a ser mayor al de ingresos. Esto también puede ocurrir en el caso de que el número de usuarios fluctúe en el tiempo.

Este caso queda reflejado en la figura 9, donde los usuarios no son constantes y en algún caso es necesario aumentar el sistema. Como se puede observar, se pueden llegar a tener grandes pérdidas, debido a que es difícil reducir el sistema al haber tenido que invertir en máquinas fisicas en propiedad.

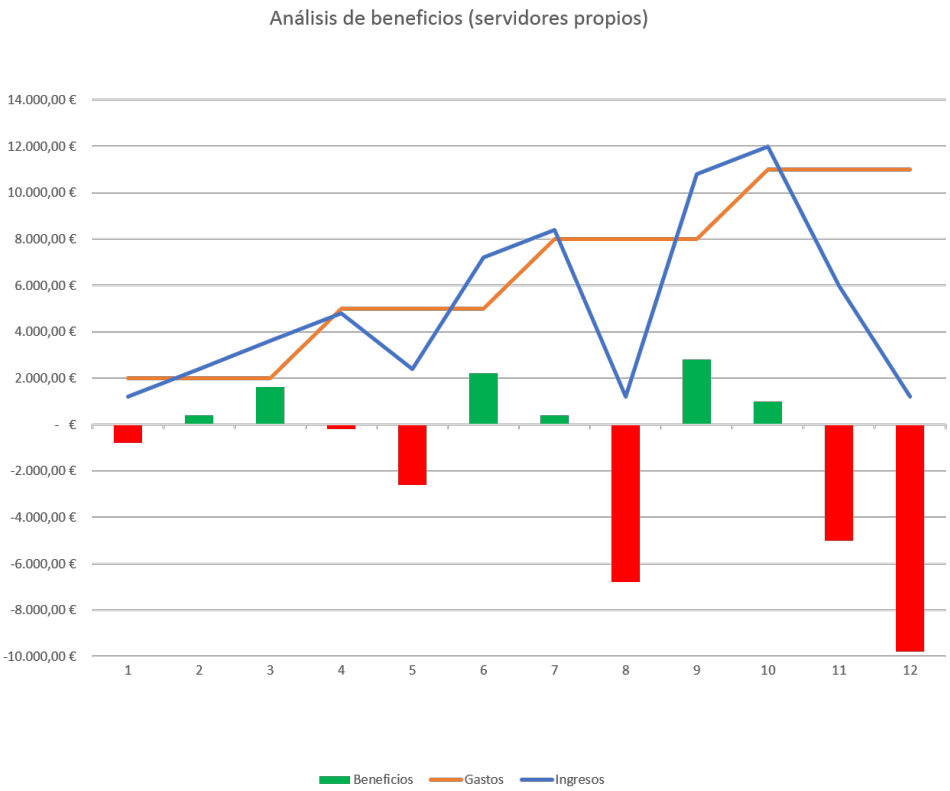


Figura 9: Gráfica de beneficios con servidores propios (pesimista).

En este ejemplo, el número de usuarios es variable. Puede darse el caso de que en un mes suba mucho el tráfico y por tanto sea necesario aumentar el sistema. El problema que supone esto es que si en un mes siguiente el número de usuarios disminuye, se puede entrar en pérdidas.

## 21.2. Sistema con servidores en *cloud*

En el caso de tener los servidores en *cloud*, si el número de usuarios aumenta, se puede ir aumentando el sistema paulatinamente, acorde a las necesidades. Además, los beneficios se mantendrán constantes.

En este caso, tal y como se puede ver en la figura 10, el valor de los ingresos con respecto al número de usuarios es menor que en el caso anterior, puesto que por lo general, el coste de los servicios cloud, ponderado y a medio-largo plazo, es mayor que el de máquinas propias.

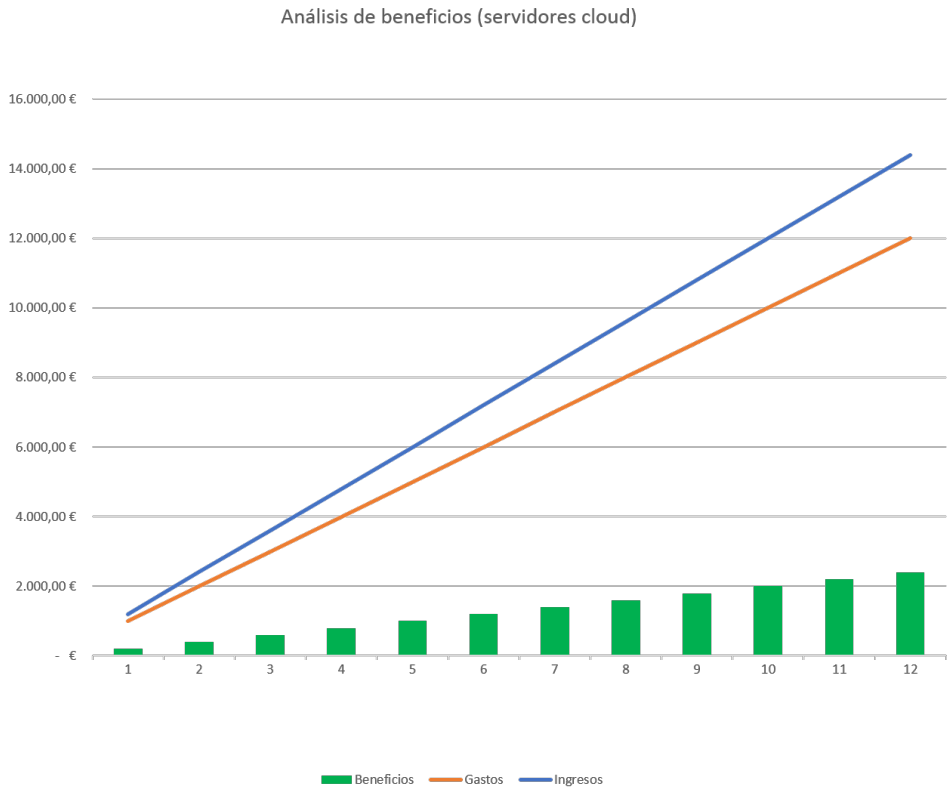


Figura 10: Gráfica de beneficios con servidores en *cloud* (optimista).

Además, si el número de usuarios fluctúa, se puede aumentar y disminuir el sistema, por lo que no es un problema. Como se puede ver en la figura 11, no se tiene por qué sufrir pérdidas (es un caso idealizado, sin tener en cuenta el número de máquinas mínimas para que el sistema pueda estar activo).

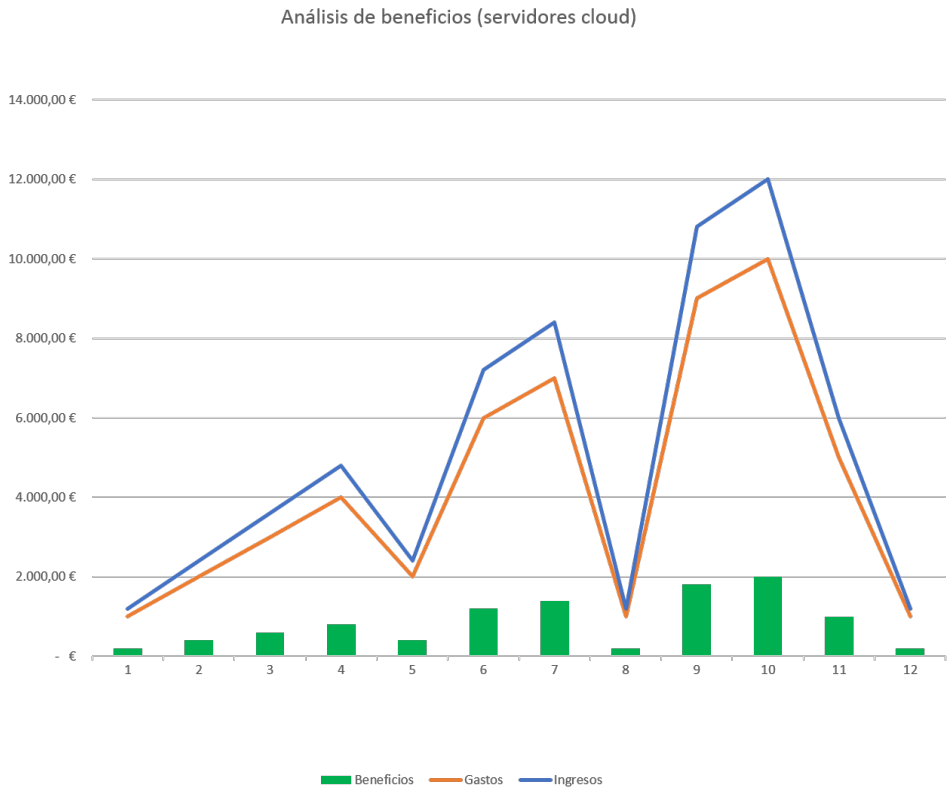


Figura 11: Gráfica de beneficios con servidores en *cloud* (pesimista).

# 22. Análisis de tráfico de entrada a los servidores web según el tiempo de caché

Este análisis trata de comprobar el beneficio que se obtiene en cuestión de carga o trabajo de los servidores web, teniendo en cuenta el funcionamiento de la caché y el número de peticiones que pueden cachearse.

Como ya hemos visto, la caché funciona como clave-valor: a cada petición determinada, le corresponde una respuesta única. Debido a esto, es necesario estudiar el comportamiento en base al número de peticiones globales, el número de peticiones diferentes, el porcentaje de peticiones cacheables y el tiempo de caché.

Con el supuesto de que haya 1.000.000 de peticiones, de las cuales el 90 % se pueden cachear, y que se subdividen en 3.000 tipos de peticiones, el número de peticiones que llegan al *backend* varía dependiendo del tiempo de caché:

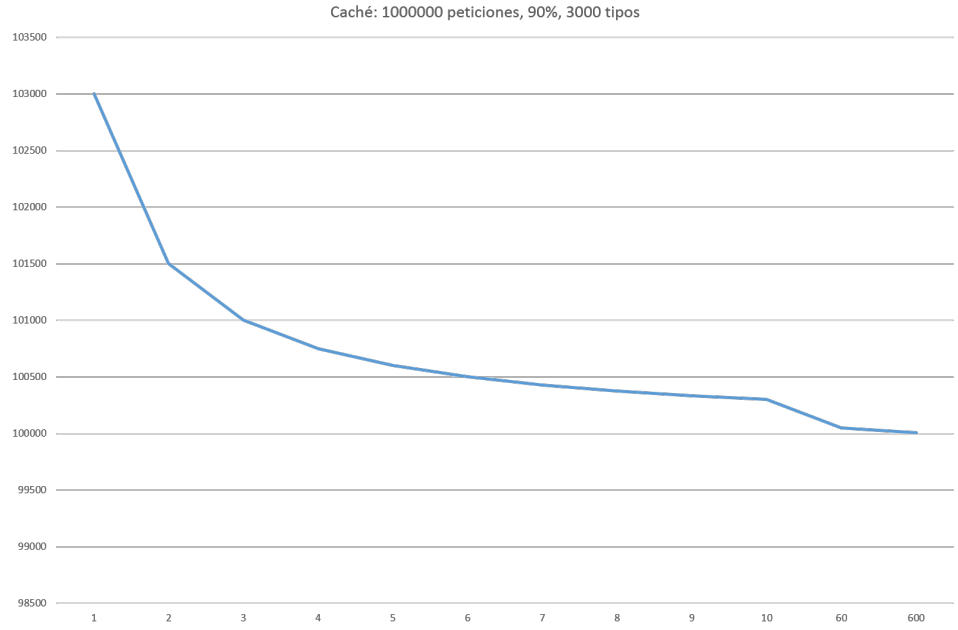


Figura 12: Gráfica del número de peticiones que llegan al *backend* después de pasar por caché.

Como se puede ver en la figura 12, el número de peticiones por segundo, según se va aumentando el tiempo de caché, converge al 10 %, puesto que sólo se tienen en cuenta las peticiones no cacheables. Al *backend* sólo le llegan las peticiones no cacheables y la primera petición de cada tipo.

En la figura 13 puede verse que las curvas basadas en el número de peticiones se comportan igual, puesto que al fin y al cabo, según el tiempo de caché, llegan al backend un porcentaje de las peticiones:

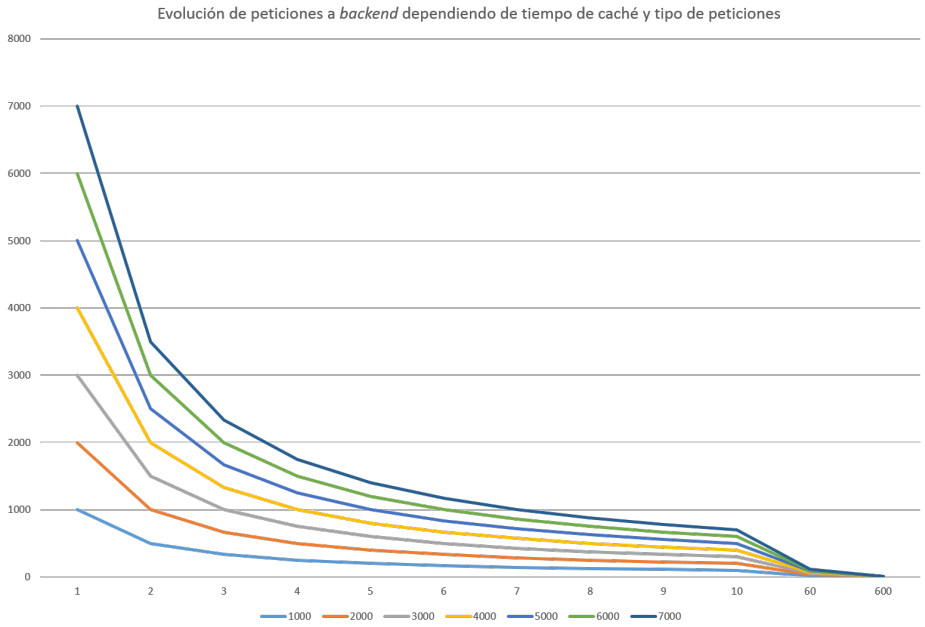


Figura 13: Gráfica del número de peticiones que llegan al *backend* dependiente de los segundos de caché.

# 23. Líneas futuras

Puesto que este trabajo no es ni solución única ni completa, consta de líneas de trabajo futuras, que ayudarían a mejorar el sistema propuesto tanto desde el punto de vista de la robustez como de la comodidad del administrador del sistema.

Estas líneas futuras no son especialmente complejas, aunque tampoco triviales. Sin duda sería una mejora que justificaría con creces el tiempo dedicado a ponerlas en funcionamiento.

## 23.1. Autoescalado

Para hacer el sistema propuesto automático y reducir costes de gestión, lo ideal sería añadir autoescalado. Esto quiere decir que el sistema podría ser capaz de saber su estado y, en caso de necesitarlo, aumentar o disminuir su número de nodos. Además, podría comprobar si algún participante del sistema está en estado de fallo, sustituyéndolo por un nodo nuevo.

Para poder llevar a cabo este autoescalado, si usamos máquinas genéricas en cloud, deberemos tener en cuenta lo siguiente:

- El proveedor debe contar con API para poder desplegar o destruir servidores.

- Debemos conectarnos a las máquinas mediante par de claves pública-privada para facilitar la gestión.
- No debemos confiarnos y se debe tener un control de los servicios.
- Por sencillez, es mejor utilizar el sistema que creemos de autoescalado con uno de orquestación.

## 23.2. Orquestación

La orquestación se basa en tener todas las configuraciones de las máquinas de modo genérico. De esta forma, desde un control central, se pueden configurar todas las máquinas por igual, discriminándolas en grupo o no, ahorrando costes, tiempo y posibilidad de fallos humanos.

De esta forma, si se necesita añadir una máquina o modificar configuraciones, es tan sencillo como modificar la plantilla y ejecutar el *script* que se encarga de conectarse a las máquinas pertinentes y reconfigurarlas.

Hay muchos programas que permiten hacer esto:

- Ansible
- Chef
- Puppet
- Salt

Es recomendable hacer un estudio de las necesidades y capacidades y encontrar el *software* que más se acerque a ellas.

# 24. Conclusiones

En este trabajo he tenido la oportunidad de investigar y aprender un gran conjunto de nuevas tecnologías orientadas a web. Gracias a esto, he podido comprender cómo funcionan los sistemas actuales, cuáles son los actores participantes en montajes complejos e incluso llegar a montar uno propio.

Debido a esto, he conseguido plasmar mis nuevos conocimientos y trabajos en una guía en la que se explican las bases de los sistemas orientados a web, impartiendo unas explicaciones previas y proporcionando los ficheros especificos de configuración para que cualquiera que lea este documento, con unas pequeñas bases, pueda montar un sistema similar que se adecúe a sus necesidades.

He podido estudiar el funcionamiento de los sistemas de orquestado, cómo funcionan y para qué sirven, pero por falta de tiempo, y con el objetivo de

no hacer apartados incompletos, no lo he incluido en esta guía pero si lo he indicado como líneas futuras, muy interesantes y necesarias, a estudiar e implementar.

Con respecto a la gestión del tiempo, he sido capaz de ir cumpliendo plazos e hitos, aunque con alguna necesidad mayor de tiempo al principio debido a las nuevas tecnologías a seleccionar y aprender.

# Parte IX
# Bibliografía

## Referencias

[1] Klaus Schmidt, *High Availability and Disaster Recovery: Concepts, Design, Implementation*. Springer, 2006 edition, 2006.

[2] Sander van Vugt, *Pro Linux High Availability Clustering*. Apress, 1 edition, 2014.

[3] Thomas Erl, *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 1 edition, 2013.

[4] Michael J. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. Wiley, 1 edition, 2014.

[5] Willy Tarreau, *HAProxy Configuration Manual*. `http://cbonte.github.io/haproxy-dconv/configuration-1.5.html`, 2014.

[6] DataStax, Inc, *Apache Cassandra$^{TM}$2.0 Documentation*. `http://www.datastax.com/documentation/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html`, 2014.

[7] Wikipedia, *High Availability*. `http://en.wikipedia.org/wiki/High_availability`, 2014.

[8] Wikipedia, *Alta disponobilidad*. `http://es.wikipedia.org/wiki/Alta_disponibilidad`, 2014.

[9] Etel Sverdlov, *How To Set Up Master Slave Replication in MySQL*. `https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql`, 2012.

[10] Planet Cassandra, *Data Replication in NoSQL Databases Explained*. `http://planetcassandra.org/data-replication-in-nosql-databases-explained/`, 2014.

[11] Ian Berry, Tony Roman, Larry Adams, J.P. Pasnak, Jimmy Conner, Reinhard Scheck, and Andreas Braun, *The Cacti Manual*. `http://www.cacti.net/downloads/docs/pdf/manual.pdf`, The Cacti Group, 2012.

[12] Wikipedia, *Simple Network Management Protocol*. `http://es.wikipedia.org/wiki/Simple_Network_Management_Protocol`, 2014.

[13] Dag-Erling Smørgrav, Poul-Henning Kamp, Kristian Lyngstøl, Per Buer, *Varnish Configuration Language*. `https://www.varnish-cache.org/docs/3.0/reference/vcl.html`, 2010.

[14] Varnish Software, *VCL Basics*. https://www.varnish-software.com/static/book/VCL_Basics.html, 2012.

[15] Per Buer, *Using pipe in Varnish*. https://www.varnish-software.com/blog/using-pipe-varnish, 2014.

[16] Jason Kurtz, *How To Set Up MySQL Master-Master Replication*. https://www.digitalocean.com/community/tutorials/how-to-set-up-mysql-master-master-replication, 2013.

[17] Oracle, *Replication Master Options and Variables*. http://dev.mysql.com/doc/refman/5.0/en/replication-options-master.html, 2014.

[18] Wikipedia, *Orchestration (Computing)*. http://en.wikipedia.org/wiki/Orchestration_(computing), 2014.

Este documento esta firmado por

| | | |
|---|---|---|
|  | **Firmante** | CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES |
| | **Fecha/Hora** | Sun Jan 04 17:40:22 CET 2015 |
| | **Emisor del Certificado** | EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES |
| | **Numero de Serie** | 630 |
| | **Metodo** | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature) |